#3

Applicant: Douglas J. Turner
Serial No. 09/809,726, Filing Date: 03/15/01
Art Unit 2661, Atty. Dkt. No. S1022/8631

PATENTS • DESIGNS • COPYRIGHT • TRADE MARKS
The Patent Office

INVESTOR IN PEOPLE

The Patent Office
Concept House
Cardiff Road
Newport
South Wales
NP10 8QQ

I, the undersigned, being an officer duly authorised in accordance with Section 74(1) and (4) of the Deregulation & Contracting Out Act 1994, to sign and issue certificates on behalf of the Comptroller-General, hereby certify that annexed hereto is a true copy of the documents as originally filed in connection with the patent application identified therein.

In accordance with the Patents (Companies Re-registration) Rules 1982, if a company named in this certificate and any accompanying documents has re-registered under the Companies Act 1980 with the same name as that with which it was registered immediately before re-registration save for the substitution as, or inclusion as, the last part of the name of the words "public limited company" or their equivalents in Welsh, references to the name of the company in this certificate and any accompanying documents shall be treated as references to the name with which it is so re-registered.

In accordance with the rules, the words "public limited company" may be replaced by p.l.c., plc, P.L.C. or PLC.

Re-registration under the Companies Act does not constitute a new legal entity but merely subjects the company to certain additional company law rules.

Signed

Dated 16 March 2001

Patents Form 1/77

Patents Act 1977
(Rule 16)

The Patent Office

The Patent Office
Cardiff Road
Newport
Gwent NP9 1RH

20MAR00 E522654-5 D00068
P01/7700 0.00-0006575.5

Request for grant of a patent

(See the notes on the back of this form. You can also get an explanatory leaflet from the Patent Office to help you fill in this form)

1. Your reference — 10143/VRD/AEM

2. Patent application number *(The Patent Office will fill in this part)* — **0006575.5** — **17 MAR 2000**

3. Full name, address and postcode of the or of each applicant *(underline all surnames)*

STMicroelectronics Limited
1000 Aztec West
Almondsbury
Bristol BS32 4SQ

Patents ADP number *(if you know it)* 7789100001

If the applicant is a corporate body, give the country/state of its incorporation — United Kingdom

4. Title of the invention — PACKET CONVERSION

5. Name of your agent *(if you have one)* — Page White & Farrer

"Address for service" in the United Kingdom to which all correspondence should be sent *(including the postcode)*

54 Doughty Street
London WC1N 2LS

Patents ADP number *(if you know it)* 1255003

6. If you are declaring priority from one or more earlier patent applications, give the country and the date of filing of the or of each of these earlier applications and *(if you know it)* the or each application number

Country	Priority application number *(if you know it)*	Date of filing *(day / month / year)*

7. If this application is divided or otherwise derived from an earlier UK application, give the number and the filing date of the earlier application

Number of earlier application	Date of filing *(day / month / year)*

8. Is a statement of inventorship and of right to grant of a patent required in support of this request? *(Answer 'Yes' if:*
a) any applicant named in part 3 is not an inventor, or
b) there is an inventor who is not named as an applicant, or
c) any named applicant is a corporate body
See note (d))

Yes

PACKET CONVERSION

The present invention relates to packet conversion and in particular to a method and system for converting packets.

It is known in the art that high performance Central Processing Units (CPUs) commonly require data which they are accessing to be held in memory in a certain alignment structure. This means that they are only able to access native multibyte data if it is held in memory at addresses which have an alignment consistent with the size of the data. For example, a given CPU will only be able to access 4 byte words if they are held in memory starting on 4 byte boundaries. If the data is not held in the correct format, when the CPU attempts to access the data, an exception is caused.

Such data can be for example, code for execution by a computer such as an instruction sequence. One particular example is a standard 32-bit processor accessing 4 byte instructions, or code which contains at least some instructions of 4 bytes in length. It will be appreciated that the word data used herein defines any bit sequences, including instructions and/or operands. Such data can only be accessed if it is stored on 4 byte boundaries. Therefore, if a data structure is mixed, that is it contains words of 4 bytes and shorter lengths, the shorter words are "padded out" with spaces of unused memory in order to ensure that the following word begins on a 4 byte boundary. Commonly, such data will have originated from a C-language compiler, which automatically inserts the appropriate spaces of unused memory in anticipation of the data being accessed by a CPU.

It is thus apparent that the inserted spaces constitute a wastage of memory space. In addition, a problem is created if the data is to be accessed by a CPU of a host computer and transmitted in packets to a different, receiving machine, since the spaces of unused memory take up valuable bandwidth. To avoid this, the host computer removes these spaces when segmenting the data into packets, so that packets are transmitted without any unused memory spaces which were present in memory merely for alignment purposes, and the receiving machine re-pads the data so that its CPU is able to read it.

format; identifying the type of packet from the type indicator; accessing the stored table for the type of packet identified and thus obtaining for each data field a value representative of a storage requirement in memory and a field descriptor for that field; and using the value and the field descriptor to load the packet into a target memory according to the target format specified by the field descriptor.

The invention will now be described, by way of example only, with reference to the accompanying drawings in which :

Figure 1 shows a packet of data ready for transmission;

Figure 2 shows a table corresponding to the packet of figure 1;

Figure 3 shows the packet of data of figure 1, together with a diagrammatic indication of how it is stored in memory;

Figure 4 is a diagrammatic representation of equipment on which the invention can be implemented; and

Figure 5 is a diagrammatic representation of a converter used in association with storage in memory of the packet of figure 1.

In the figures, like reference numerals indicate like parts.

The situation in which a preferred embodiment of the present invention operates is when a first, transmitting machine needs to transmit data which it must access from a memory where it has been stored by a C-language compiler. In this case, the data is in the form of packets, each containing a number of words, some of which words are of four bytes in length. Each packet can be one of a number of different types, but none of them contains any words longer than 4 bytes, although it is likely that they will contain some words of a shorter length. They may also contain nested words. Furthermore, the C-language compiler will have arranged the words such that no words are stored over a four-byte boundary, by leaving unused spaces in memory after words which are shorter than 4 bytes, such that the following word begins on a four-byte boundary. In other words, the data that the transmitting machine accesses has been aligned by the C-language compiler. The words and unused spaces are collectively known as the fields of the packet.

length, as shown by bracket 6. The next byte is an integer word of one byte in length, as shown by bracket 8. The remaining five bytes are a nested data structure, as shown by bracket 14. Within this nested data structure 14, the first byte is an integer word of one byte in length, as shown by bracket 10, and the remaining four bytes are an integer word of four bytes in length, as shown by bracket 12.

Thus in written form, the packet looks as follows:

```
{
  4-byte integer word
  4-byte text word
  1-byte integer word
        {
          1-byte integer word
          4-byte integer word
        }
}
```

Referring next to figure 4, there is shown a system comprising a transmitting machine 102 and a receiving machine 104. There is also shown a connection 114 between the transmitting machine 102 and the receiving machine 104, along which data is being transmitted from the transmitting machine 102 to the receiving machine 104. This data is in the form of packets 116, some of which will be packets of the type of packet 1 shown in figure 1.

Within transmitting machine 102 there is a memory 106, from which data is received by a converter 108. The converter also receives information from a table store 110. Table store 110 stores a number of tables for use in data conversion, one for each known type of packet, as will be described in detail below. Having converted the data (as described below), the converter 108 sends it to an output port 112 from which it is transmitted along the connection 114.

The receiving machine 104 has an input port 118 which receives transmitted data from the connection 114 and sends it to a converter 122. The converter 122 is provided with a two-way connection with each of a table store 120 and a memory 50.

conversion memory 140, which holds a swap function routine pointed to by a swap function pointer 142 and a copy function routine pointed to by a copy function pointer 144.

In operation, the packets from input port 118 of receiving machine 104 (see figure 4) are received into packet buffer 130. Source pointer 132 identifies the first word of a packet to be stored and destination pointer 124 points to an area within memory 50 in which this word is to be stored. A data conversion is carried out on each word before it is stored in memory 50, and after each storage, the two pointers 132, 134 will move accordingly to the next position in the packet or memory 50 accordingly. The exact way in which these movements are dictated will be described in detail below with reference to figure 3.

In order to allow the correct conversion of data within each packet, the header reader 136 reads the packet header to identify the type of packet. For example, it may identify a packet of the type 1 shown in figure 1. In this case the endianess of the packets will be MSB format. This information is passed to the conversion process 138. The conversion process identifies the receiving machine as LSB and so points to the swap function routine in the conversion memory 140. If the receiving machine was MSB, i.e. the same as the transmitter, the pointer 144 would be used to point to the copy function machine. If the swap function routine is set, it is important to note that not all data is swapped, as described later. That is, byte oriented data (e.g. ASCII text) is copied while native multibyte data types (e.g. integer values) are byte swapped, according to the field descriptor.

For the moment, we will assume that the transmitting machine has successfully converted the data and transmitted it without the memory spaces added by the C-language compiler, therefore the following will describe how the receiving machine deals with the incoming data. Further explanation of the transmission process will be given subsequently.

The CPU of the receiving machine is not interested in the format in which the data is received, providing it can access the data in a format suitable for it. Therefore, the conversion of the data from the received format to the CPU-accessible format can be done by the receiving machine independently from the CPU.

multibyte value, bits 0-29 indicate the length of the field as well as inferring a required alignment. If the field is byte oriented data (e.g. ASCII text), bits 0-29 merely indicate the length of the field, but do not infer an alignment. The third possibility is for bits 0-29 to merely indicate an alignment requirement ; in this case, the actual alignment required for the field is not the same as would be expected for the size of the field represented by the subsequent row if there were no alignment requirements. In other words, this type of alignment requirement is used to ensure that the next word is correctly aligned.

The possible values of the field descriptors, bits 31 and 30, and the meaning of the remaining bits, bits 0-29 can be summarised by consulting the following table :

Bit 31	Bit 30	Meaning
0	0	Normal data field, subject to byte swapping when packing/unpacking. Bits 0-29 indicate the length of the field and provide the implicit alignment required for the field.
0	1	Normal data field, however its contents should not be byte swapped when packing/unpacking. Bits 0-29 indicate the length of the field but do not infer a required alignment for the field.
1	0	This specifies an alignment requirement. No data is copied for entries of this type, however the associated data pointer is adjusted forward to comply with the alignment specified in bits 0-29.
1	1	Not supported.

The specific values in the table of figure 2 will now be explained with reference to the data packet of figure 1. The rows of the table are labeled alphabetically for convenience of explanation.

Row A is for the 4-byte integer word 4. Therefore, within bits 0-29 there is an indication that 4 bytes of memory space will be required for this word, and that it should be aligned on an address boundary suitable for a 4 byte data type on the CPU 124, as would be expected from its length. The field descriptor has a value 0 0, because word 4 is an integer, which means that it may require an endianess swap.

Finally, the last row of the table, row G, specifies a value of 0, indicating the end of the table.

The way in which the receiving machine uses the information in the table will now be described with reference to figure 3.

Figure 3 shows the data packet 1 of figure 1. Reference numeral 50 indicates generally a diagrammatic representation of memory in the receiving machine, divided into 1 byte areas. There are also shown in various locations, the source pointer 132, pointing towards word boundaries of packet 1 and the destination pointer 134, pointing towards areas of memory 50.

Upon accessing table 20, the converter 122 of receiving machine 104 begins at row A and locates the start of the first word, 4-byte integer word 4, and moves source pointer 132 to indicate this start, shown diagrammatically at position 60. The word 4 is then processed through an endianess-swapping routine, as indicated by the swapping function pointer 142, which was set as described above in order to change its endianess, and then stored in a 4-byte area of memory 80, the start of which is indicated by destination pointer 134, shown diagrammatically at position 160.

Since the first row of table 20 has the field descriptor 0 0 and because the length is indicated as 4 bytes in bits 0-29, the source pointer 132 is then moved to position 62, the start of the next word 6. The value of 4 in table 20 implies an alignment requirement on a 4 byte boundary, therefore the destination pointer 134 is moved to position 162, the start of the next available area of memory 82, which is also on a 4 byte boundary.

The converter 122 then reads row B of table 20 and processes the next word, 4-byte text word 6. Its field descriptor 0 1 means that it is not passed through the endianess-swapping routine, but instead is copied directly via copy function pointer 144 into the next 4-byte area of memory 82, the start of which is indicated by destination pointer 134 at position 162.

whose start is indicated by the position 172 of destination pointer 134, bracketed as 92.

Subsequently-received data packets can then be stored in memory, if desired beginning at position 174 at the end of area 92 of memory, which is on a 4-byte boundary.

It will be appreciated that the above-described method would be applicable to other data packets having different arrangements of words. Such other data packets could have a structure such that the maximum word length is four bytes, as in the packet 1, or they could have a different maximum word length. In all cases, the alignment requirement is dictated by the longest word (including within any nested words). For example, if the maximum word length were 8 bytes, the data would need to be stored on the appropriate boundaries in memory as required by the machine. This might be on 8 byte boundaries, for example as required by a 64-bit machine, or alternatively, on some 32-bit machines, it would be on 4 byte boundaries.

The equivalent data transmission process can now be explained in relation to the receiving process discussed. In the transmitting machine, the data of packet 1 will be stored in memory 106 in a similar way as it is shown to be stored in memory 50 of figure 3. This means it will be stored with unused memory similar to unused memory 86. In order to transmit packet 1, the converter 108 of transmitting machine 102 must process the data to produce packet 1. This processing can in fact be done using the reverse process to the receiving process. This means that source pointers will move along the stored data in a similar way to the destination pointers 134 of figure 3, taking account of any unused memory spaces, and destination pointers will move in a similar way to source pointers 132, in order to place the data in packet format, without any unused spaces. This means that the same table 20 can be used as is used for the receiving process. The endianess of the data could be changed if necessary, should this be desirable prior to transmission. The final stage would be to add the header 2, which would include a type identifier to enable the receiving machine to access the correct table.

The creation of the tables will now be discussed.

CLAIMS:

1. A method of converting a packet of data from a source format to a target format, the packet comprising a type indicator and at least one data field, the method comprising the steps of :

storing a table for each packet type, each table comprising for each data field of that packet type a value representative of a storage requirement in memory and a corresponding field descriptor denoting the nature of the data field;

receiving a packet in a source format;

identifying the type of packet from the type indicator;

accessing the stored table for the type of packet identified and thus obtaining for each data field a value representative of a storage requirement in memory and a field descriptor for that field; and

using the value and the field descriptor to load the packet into a target memory according to the target format specified by the field descriptor.

2. A method according to claim 1, wherein the field descriptor denotes the nature of the data field as one of a used field and an unused field.

3. A method according to claim 2, wherein the field descriptor denotes the nature of a used field as one of a text field and an integer field.

4. A method according to claim 2 or claim 3, comprising the further steps of :

using a source pointer to identify each data field in the received packet;

determining whether the data field is used or unused;

where the data field is used, using a target pointer to identify a location in memory based on the value representative of the storage requirement in memory of that field and storing the field in the identified location in memory;

where the data field is unused, using a target pointer to identify a location in memory based on the value representative of the storage requirement in memory of that field and moving the target pointer to a new location in memory corresponding to the end of the storage requirement.

descriptor to load the packet into the target memory according to the target format specified by the field descriptor.

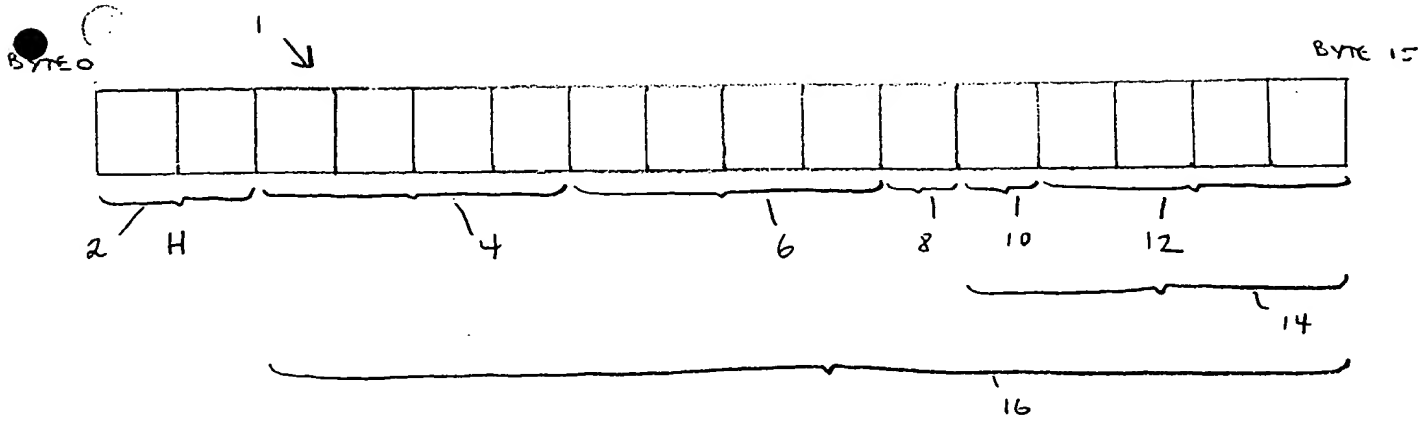

FIGURE 1

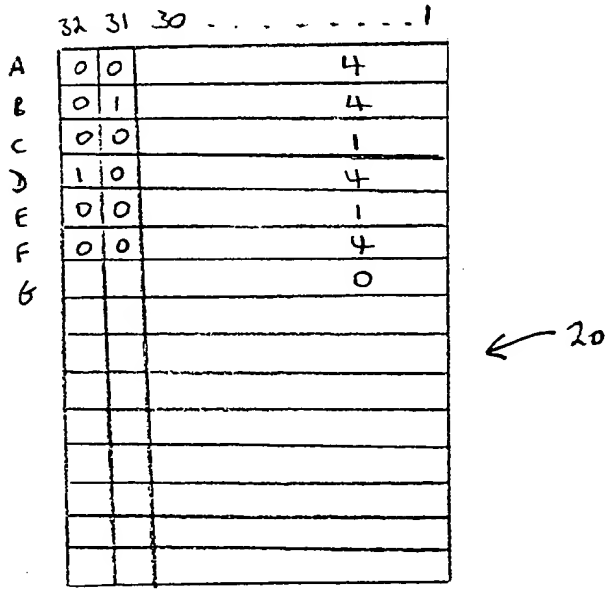

FIGURE 2

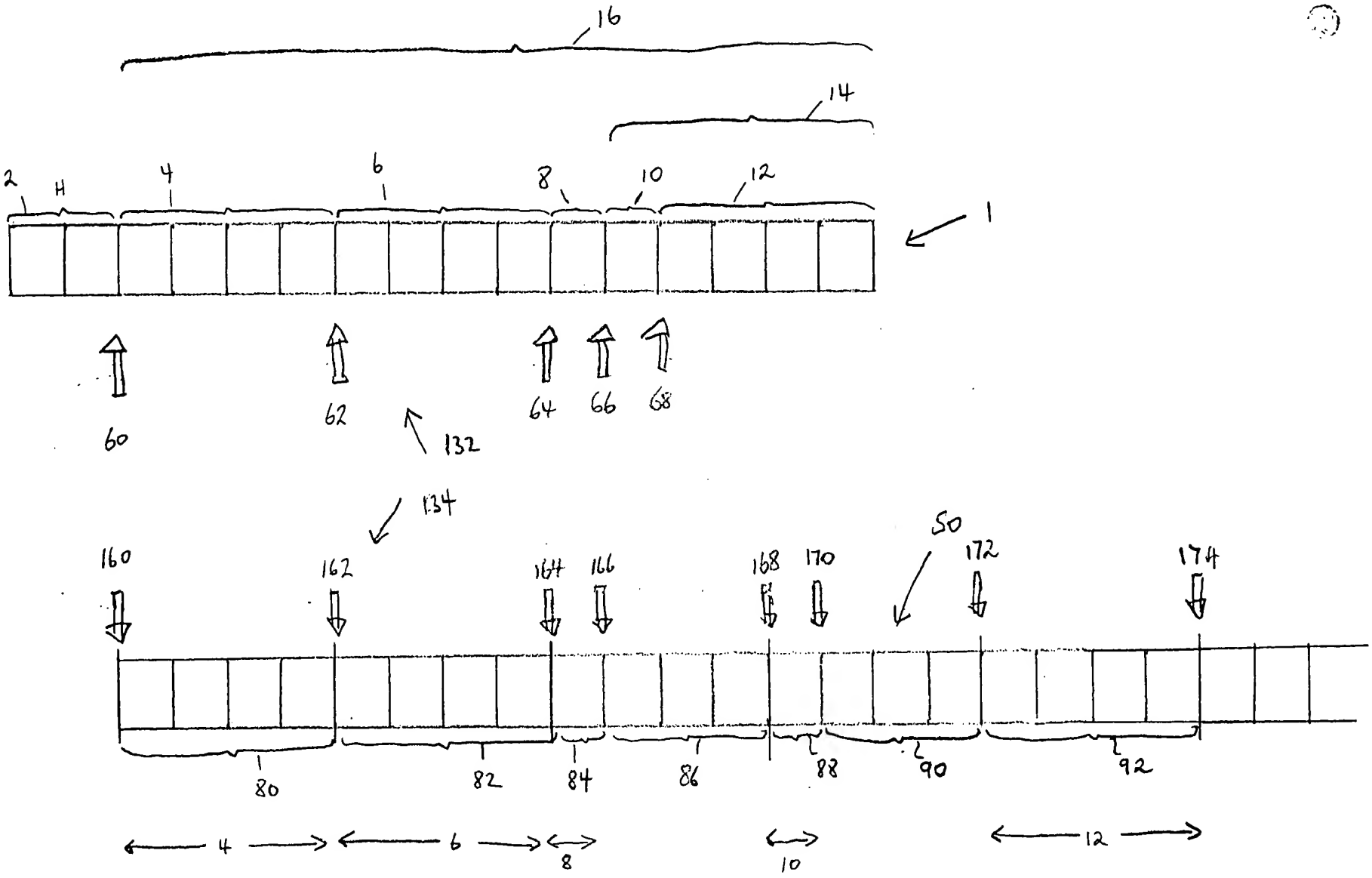
16
14
2
H
4
6
8
10
12
1
60
62
132
64
66
68
134
160
162
164
166
168
170
50
172
174
80
82
84
86
88
90
92
4
6
8
10
12

FIGURE 3

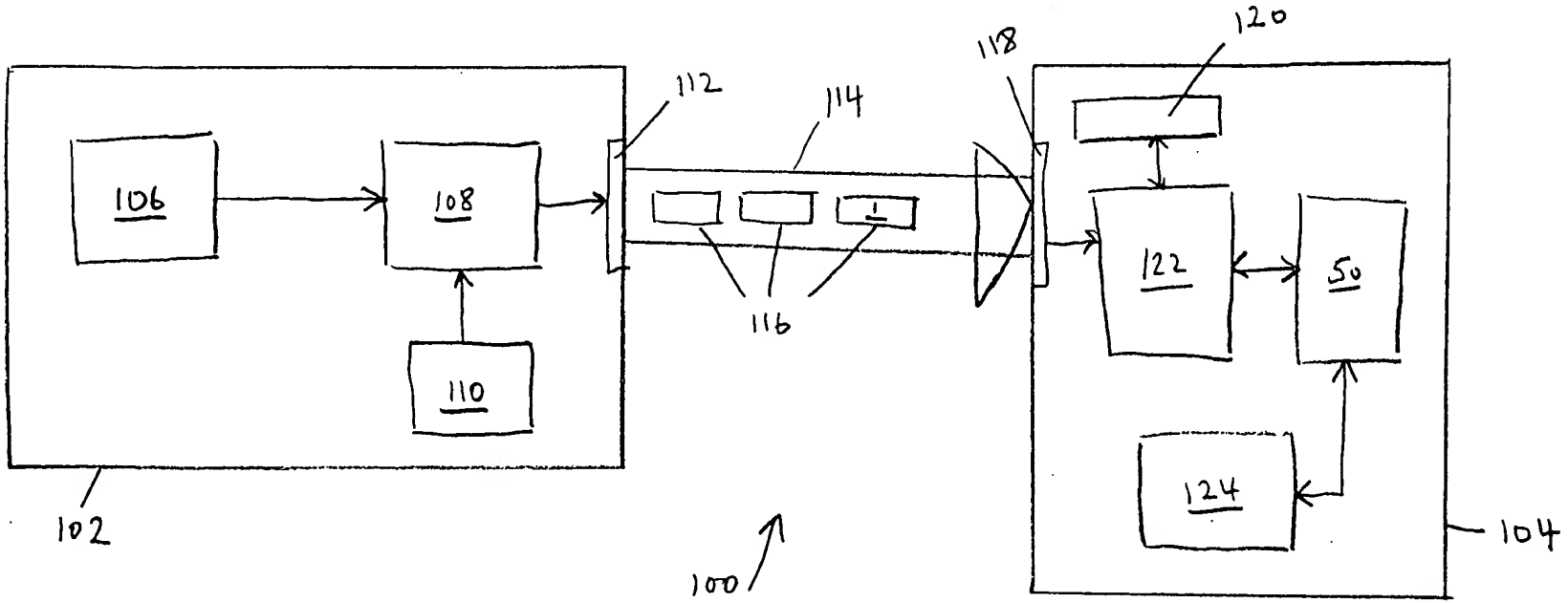

FIGURE 4

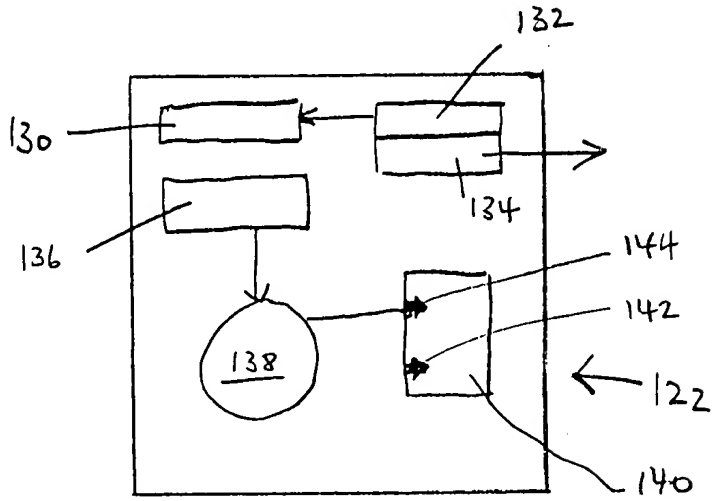

FIGURE 5